Programming Assignment #2: NS-3 Simulator

University at Buffalo-Spring 2025

CSE 4/589 Modern Networking Concepts

Project Overview

This project is designed to review concepts of the CSE4/589: Modern Networking Concepts course such as routing, congestion control, delays, and network quality of Service (QoS) parameters through the NS-3 simulation tool. NS-3 is a powerful discrete-event network simulator for Internet systems, targeted primarily for research and educational use. NS-3 is free, open-source software, licensed under the GNU GPLv2 license. You can find more details about this tool on its website and comprehensive tutorial file.

Deliverables

Programming assignment #2 includes four tasks and a bonus task with different deliverables. You will find more details about the deliverables of each task in the project task section. You should submit your assignment as a zip file containing the following items to Brightspace.

- Simulated file: A C++ file or a Python file as your simulated scenario through NS-3.
- Project report: A PDF file as a report showing your simulation results and discussions.
- Project demo: A video file showing your project presentation.

You can find details of each item in the following sections. This is optional and up to you if we have not provided instructions or guidelines about them on this file such as deliverables, page limitations, report structure, and simulation parameters.

Notes:

- We will run your simulated project on our computer to validate your project and check the similarity of your code. Therefore, you should highlight your NS-3 version and Ubuntu version in your project report file.
- Your report should be typed and cover all requested concepts, handwritten assignments will not be accepted.
- In the project demo, you should show Task 1 and Task 4 running on your system. Record a 15-20-minute video and present your work. Each team member should participate in the presentation process, your webcam must be turned on, and each presenter must first introduce themselves.
-

General Notes:

- You should adhere to Academic Integrity. You will get an "F" for the course if you violate the academic integrity. See the course syllabus for more details.
- Team members are equally responsible for any AI violation. Therefore, do not let your teammate violate AI policy.
- For individual assignments, students who share their work are equally responsible for AI violations as those receiving the material.
- See the course syllabus for the late policy of PA#2.
- Programming assignment #2 can be conducted either individually or with a maximum of two students.
- Although it is recommended to install the latest version of NS-3 (NS-3.43) and Ubuntu 24.04.01, you can also install older versions of NS-3, such as NS-3.42 or NS-3.41, and older version of Ubuntu such as 22.04. It should be noted that you cannot conduct PA #2 through NS-2 for the reasons given on page 12 of the tutorial file.
- Member #1 must submit the report, one submission per team.
- Your teammate in this programming assignment may be different than your teammate in PA#1. Moreover, undergraduate and graduate students can work jointly but, in this case, you should meet the deadline of graduate teams.
- Name all files with team members' UBIT such as UBITmember1_UBITmember2.zip.
- PA#2 will be graded out of 100. Bonus task adds 10 extra points to your PA2's grade.

NS-3 Installation Guidelines

NS-3 can be installed on Linux, FreeBSD, and MacOS operating systems. According to Table 1 of the installation guidelines, some features of ns-3 are not supported by FreeBSD and MacOS, therefore, it is better to install it on Linux. You can find installation guidelines in the following links.

1) **How to install ns3 in Ubuntu | ns-3.41 in Ubuntu 22.04**

Note: The provided steps in the above video can be utilized to install the latest version of NS-3.43 (released on October 9, 2024) on the latest version of Ubuntu 24.04.01 (released on August 29, 2024). At the time of writing this file, ns-3.42 was installed on my system, so you will see this version in the screenshots and commands, which can be replaced by another version on your system.

2) **ns-3 Installation Guide**

In case of using MacOS or Windows, you need first to install Ubuntu on your system. You could find many online resources, such as

How To Install Ubuntu On M1 Mac

How to Install Ubuntu on VirtualBox in Windows

In general, you can find more details about the installation of NS-3 on Windows or MacOS in the following link in case of any installation problems.

Installation Guidelines for macOS and Windows

After installation, as explained in the installation video and on page 45 of the tutorial file, make sure that the shared library paths are set correctly and that the libraries are available at run time by running the following scripts. Therefore, you can check the working of ns-3 on your system by running a few simple scripts located on the "~ns-3.42/examples/tutorial/" directory such as *first.cc, second.cc*, and *hello-simlator.cc*.

In general, *$ ./ns3 run <ns-filename>* command can be utilized to run a ns-3 code. This command should be run in the "~/ns-3.42" directory because ns-3 has been installed in this directory. Moreover, in this command, <ns-filename> should be written without any extension, such as .cc or .py, as you see in the following examples. For example, to run the first.cc script just write *first* instead of *first.cc*.

$ ./ns3 run first

You will get the results as follows. You can find explanations of these outputs and walkthrough for the first.cc script on page 59 and on page 51 of the tutorial file, respectively.

```
shahram@shahram:~/ns-allinone-3.42/ns-3.42$ ./ns3 run first
[0/2] Re-checking globbed directories...
ninja: no work to do.
At time +2s client sent 1024 bytes to 10.1.1.2 port 9
At time +2.00369s server received 1024 bytes from 10.1.1.1 port 49153
At time +2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
At time +2.00737s client received 1024 bytes from 10.1.1.2 port 9
```

Note: In this command, just the name of the file (*first*) is written because there is only one ns-file entitled "*first*" in ~/ns-3.42 directory, if there are several codes with the same name in different sub-directories you should clarify the sub-directory of the file in your command. For example, $ ./ns3 run scratch/first or $ ./ns3 run examples/tutorial/first

$ ./ns3 run second

You will get the results as follows. You can find explanations of these outputs and walkthrough for the second script on page 81 and on page 77 of the tutorial file, respectively.

```
shahram@shahram:~/ns-allinone-3.42/ns-3.42$ ./ns3 run second
[0/2] Re-checking globbed directories...
ninja: no work to do.
At time +2s client sent 1024 bytes to 10.1.2.4 port 9
At time +2.0118s server received 1024 bytes from 10.1.1.1 port 49153
At time +2.0118s server sent 1024 bytes to 10.1.1.1 port 49153
At time +2.02161s client received 1024 bytes from 10.1.2.4 port 9
```

$ ./ns3 run hello-simulator

You should get the results as follows.

```
shahram@shahram:~/ns-allinone-3.42/ns-3.42$ ./ns3 run hello-simulator
[0/2] Re-checking globbed directories...
ninja: no work to do.
Hello Simulator
```

If you have got similar outputs without any error, it means that ns-3 libraries and required tools are installed on your system. If you did not get the desired outputs, you can refer to page 44 of the tutorial file, or as a suggestion, run the following command to enable the examples.

$ ./ns3 configure --enable-examples --enable-tests

Then run the following command to build ns3.

$ ./ns3 build

Note: There is a Phyton-based test script entitled “test.py” in the “~/ns-3.42” directory that can be used to check installed functions and libraries and their configurations. You can run it through $./test.py command and check whether all components pass or fail. You should get results as follows.

```
[765/769] PASS: Example src/energy/examples/rv-battery-model-test
[766/769] PASS: Example src/bridge/examples/csma-bridge
[767/769] PASS: Example src/bridge/examples/csma-bridge-one-hop
[768/769] PASS: Example src/aodv/examples/aodv
769 of 769 tests passed (769 passed, 0 skipped, 0 failed, 0 crashed, 0 valgrind errors)
```

The status of each component is represented on each line, and the last line of the output confirms that all 769 components are correctly installed and configured on the system. The number of components can vary on your system, but they must all be passed correctly.

You evaluated installed libraries and their configurations through three simple scripts, i.e., first.cc, second.cc, and hello-simulator.cc. These scripts are located in the example/tutorial directory, also example directory consists of several sub-directories such as ipv6, routing, wireless, and socket including a lot of useful ns-3 codes. You can open each script and analyze

its code in your favorite editor such as Visual Studio or gedit. You can find explanations of all built-in functions, classes, and APIs of ns-3 in its Doxygen.

Note: If you want to analyze/change a script such as first.cc, due to two reasons it is essential to copy it to "~/ns-3.42/scratch/" directory and work on the copied file. Firstly, the original code in the "~/examples/tutorial/" directory will be kept safe, as explained on page 59 of the tutorial file. Secondly, you cannot compile some original code in the tutorial directory if you add some new classes or functions.

Visualization of Projects

In the previous section, you were able to run the two simple scripts and get the simulation result as text printed on your screen. Moreover, you can visualize the simulated scenarios through *NetAnim* as a visualizer tool.

For example, first.cc script just simulated a point-to-point network including two nodes, which its simple topology is presented at the beginning of its source code. You can follow the following steps (on first.cc script) to visualize it through *NetAnim*.

Step 1: Modification on the first.cc to generate an XML file.

In this step, you must add a few simple snippets to the first.cc to create an XML file that can be opened through NetAnim. Firstly, add #include "ns/netanim-module.h" to the includes statements section. Then, add the following snippet exactly before the simulator function (Simulator::Run();).

```
AnimationInterface anim ("first1.xml");
anim.SetConstantPosition (nodes.Get(0), 10.0, 10.0);
anim.SetConstantPosition (nodes.Get(1), 2.0, 20.0);
```

The first line declares the name of the XML file that will be generated. The second two lines declare the positions of two network nodes. According to this snippet, node 0, the client of this scenario, will be located at (x=10, y=10) and node 1, the server of this scenario, will be located at (x=2, y=20).

The "first1.xml" will be created in the "…/ns-3.42" directory if you run the code as follows. Note: As you see, I copied first.cc to the scratch directory, renamed it to first1.cc, and added the mentioned pieces of code to it.

```
shahram@shahram:~/ns-allinone-3.42/ns-3.42$ ./ns3 run scratch/first1
[0/2] Re-checking globbed directories...
ninja: no work to do.
AnimationInterface WARNING:Node:0 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:1 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:0 Does not have a mobility model. Use SetConstantPosition if it is stationary
AnimationInterface WARNING:Node:1 Does not have a mobility model. Use SetConstantPosition if it is stationary
At time +2s client sent 1024 bytes to 10.1.1.2 port 9
At time +2.00369s server received 1024 bytes from 10.1.1.1 port 49153
At time +2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
At time +2.00737s client received 1024 bytes from 10.1.1.2 port 9
```

Note: The printed result includes two parts. The first four lines confirm that an XML file has been created, if you do not have the same output on your screen, it means that you need to check the code and modules installed on your system. It should be noted that these four lines are warnings and do not affect your execution. The next four lines clarify four different events of your simulation.

If you have a problem in the creation of .xml file or in opening it by NetAnim, as a suggestion check/install the following libraries If you are sure that everything is in order.

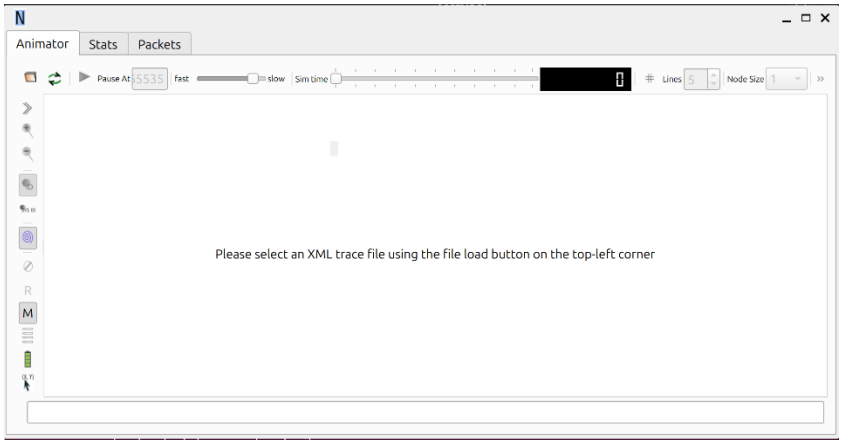
$ sudo apt install libxml2

$ sudo apt install libxcb-cursor-dev

Moreover, you can check available updates through the *$ sudo apt update* or *$ sudo apt-get update* commands, and if any modules are not installed, upgrade your system with *$ sudo apt upgrade* command.

Step 2: Executing NetAnim

NetAnim is a visualizer tool that is automatically installed through the ns-allinone-3.42 package. You can find installed NetAnim in the "~/ns-allinone-3.42/netanim-3.109" directory. Run it through $ ./NetAnim command in the mentioned directory and you will see the NetAnim tool in the following figure.



Step 3: Opening the XML file through NetAnim

You can open (top-left icon) the generated xml file (first1.xml in this example) and run it through the play icon and see the simulation scenario. At first, the network nodes are placed randomly in the grid area, but they will be relocated to the determined positions after playing the simulation. As a simple graphic tool, you can do simple settings such as zoom-in, zoom-out, adjusting simulation speed, changing node size, and checking transmission of packets at different time stamps, also you can check the statistics of nodes under *stats* tab, and transmission process over time under *packets* tab.

Simulation analysis through TraceMetrics

TraceMetrics is an open-source and Java-based tool for analyzing the trace files that can be generated in the simulation of ns-3 codes. Simple scenarios usually do not produce extensive results and can be easily evaluated and analyzed, while for the evaluation of complicated scenarios, you need an effective tool to classify and sort the simulation results. TraceMetrics is an effective graphical tool for this purpose to analyze complicated simulation results, calculate useful metrics, and quick performance measurements. You can follow the following steps (on first.cc script) to analyze the simulation results through TraceMetrics.

Step 1: Modification on the first.cc to generate a trace file.

You can add two following commands before the simulator function (Simulator::Run();) to create a trace file including logs in the Ascii format.

```
AsciiTraceHelper ascii;
pointToPoint.EnableAsciiAll(ascii.CreateFileStream("first2.tr"));
```

The first line of code creates an object entitled *ascii* of type *AsciiTraceHelper* class. The second line enables the creation of a trace file on the point-to-point channel between client and server to generate a trace file entitled “first2.tr”.

Now, if you run the code you can see a trace file entitled *first2.tr* in the “~/ns-3.42” directory. You can open it through text editor tools and see the simulation results in ASCII format. You will see a lot of details about the simulation results of first.cc and you will need an interpreter to parse this ASCII information and display it in an appropriate style.

Step 2: Install TraceMetrics to analyze the trace file.

You can download the latest version of TraceMetrics, tracemetrics-1.4.0, through this link. Once downloaded, copy it to the Ubuntu root, unzip it, and you will see tracemetrics.jar, an execution file, in the tracemetrics-1.4.0 directory. This is a Java-based file, you can install it through the following command if you do not have it on your system

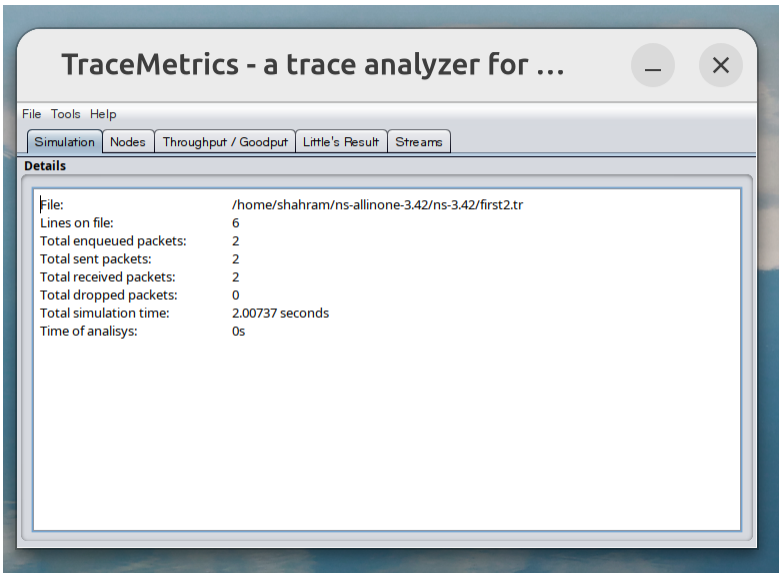
$ sudo apt install default-jdk

Step 3: Open TraceMetrics and analyze the trace file.

You can open TraceMetrics through the following command that should be run in the ~/tracemetrics-1.4.0/ directory.

$ java -jar tracemetrics.jar

You can select and open the trace file and get classified information about the overall simulation results, network nodes, throughput, and UDP streams.



Simulation analysis through Wireshark

There is another way to create trace files and analyze them through Wireshark. In this case, you can generate trace files in .pcap format, which pcap stands for packet capture, including all information of captured packets. The pcap format packets can be analyzed through traffic trace analyzers such as Wireshark and tcpdump. Let's take a look at the second script (second.cc) and follow the following steps to analyze it through Wireshark.

Step 1: Modification on the second.cc to create a .pcap file.

After copying it to the scratch directory, you can start analyzing or modifying it. You can find the following two commands in this script.

```
pointToPoint.EnablePcapAll("second");
csma.EnablePcap("second", csmaDevices.Get(1), true);
```

The first line of code enables the creation of a .pcap trace file for the point-to-point part of the network, recording in a file entitled *second*. The second line enables the creation of a .pcap trace file just for the node having index one, meaning the second node of the LAN part of the network, recording in the file entitled *second*. As an example, you can change the above snippet as follows to capture the packets in different files.

```
pointToPoint.EnablePcapAll("p2p");
csma.EnablePcap("csma1", csmaDevices.Get(1), true);
csma.EnablePcap("csma2", csmaDevices.Get(3), true);
```

Note: In contrast to the NetAnim and trace files, where we had to define type extension in the naming of the output file such as “first.xml” or “first.tr”, no need to add .pcap extension in packet capturing files.

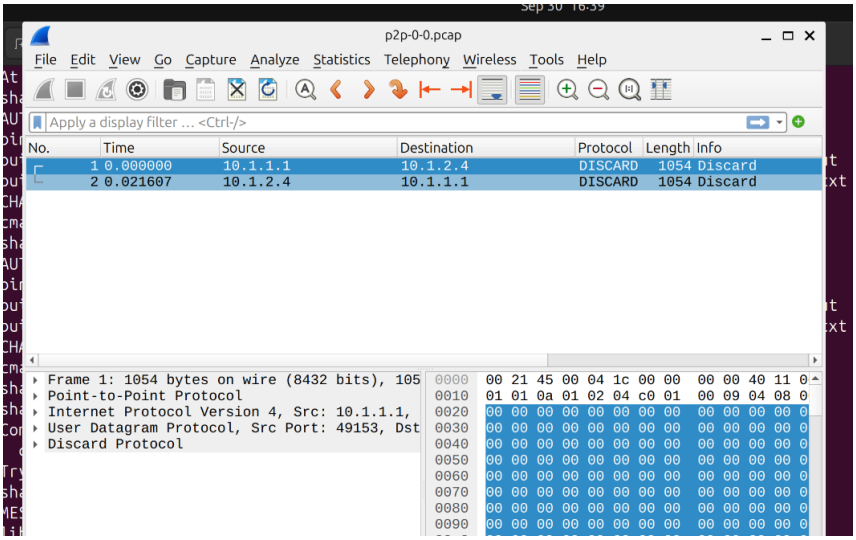
After running second.cc you can find four pcap files, i.e., p2p-0-0.pcap, p2p-1-0.pcap, csma1-2-0.pcap, and csma1-4-0.pcap corresponding to captured packets of node 0-device 0 of point-to-point part, node 1-device-0 of point-to-point part, second node of LAN part-device 0, and fourth node of LAN part-device 0, respectively. In other words, p2p-1-0 includes captured packets of device #0 (NIC #0 or first NIC) of node #1 (or second node) of the point-to-point part.

Step 2: Open Wireshark and analyze the pcap files.

Wireshark is a packet analyzer tool that is automatically installed through the ns-allinone-3.42 package. You can run it inside the “~/ns-3.42/” directory as follows.

$ ~/ns-allinone-3.42/ns-3.42/ wireshark <*pcap file name*>

You can see the Wireshark window as follows. You can analyze the captured packets like Wireshark Lab assignments.



Step 3: Utilizing tcpdump for analyzing the pcap files.

The tcpdump is another tool to analyze the pcap files. This is one of the built-in tools of ns-3 that can be run in the terminal as follows. You can find more details about the following output on page 82 of the tutorial file.

```
shahram@shahram:~/ns-allinone-3.42/ns-3.42$ tcpdump -n -t -r csma2-4-0.pcap
reading from file csma2-4-0.pcap, link-type EN10MB (Ethernet), snapshot length 65535
ARP, Request who-has 10.1.2.4 (ff:ff:ff:ff:ff:ff) tell 10.1.2.1, length 50
ARP, Reply 10.1.2.4 is-at 00:00:00:00:00:06, length 50
IP 10.1.1.1.49153 > 10.1.2.4.9: UDP, length 1024
ARP, Request who-has 10.1.2.1 (ff:ff:ff:ff:ff:ff) tell 10.1.2.4, length 50
ARP, Reply 10.1.2.1 is-at 00:00:00:00:00:03, length 50
IP 10.1.2.4.9 > 10.1.1.1.49153: UDP, length 1024
```

Creation of graph through Gnuplot

In general, graphs and charts can organize and display complex data and make it easier for people to understand. For this reason, sometimes we would like to create a graph or chart for obtained simulation results. Although you can extract the required data from Log files, TraceMetrics, or Wireshark and plot a graph by mathematical and graphical tools such as Excel or MATLAB, you can directly plot a chart through the Gnuplot tool. There are three methods to plot a graph through Gnuplot as follows. As explained in the following video, first install the gnuplot on your system and plot a few simple functions.

Method I) Manual data collection and manual plotting

You can manually extract the data from simulation results and create a data file and a code file, and then create a graph through terminal commands. You can find the guidelines in the following links.

Official gnuplot documentation

Gnuplot for plotting data - NS3 Tutorial 5

Likewise, in the following video, you can find instructions on the creation of a histogram through Gnuplot.

Histogram using Gnuplot

Method II) Automatic data collection and manual plotting

Following the instructions of the above links, you can manually create a data file, a simple code file, and plot your desired chart. This technique will be time-consuming if the simulation results are extensive, and even it is mistake-prone due to manual data collection. Therefore, you can automate data collection, although plotting is still manual. Automatic data collection or creation of a .dat file can be done through two methods.

a) Through terminal commands

Let's run the fifth.cc script and see the results on the screen. You will get a lot of printed messages on your screen if you run fifth.cc. The printed outputs show the new size of congestion window (cwnd), the moment of changing cwnd, and the moment of dropping packets (RxDrop). You can easily record these outputs in a .dat file through the following terminal command and adding options to the run command.

$ ./ns3 run fifth > <file name>.dat 2>&1

This command creates a .dat file inside the ~/ns-3.42/ directory with the <file name> you have defined in the above command that consists of all the printed outputs, you can check it by a text editor tool. It should be noted that the data file is created automatically, but if you want to plot it through Gnuplot you need to run several commands in the Gnuplot environment, as you see on page 127 of the tutorial file.

b) Through AsciiTraceHelper and PcapHelper

You are familiar with AsciiTraceHelper and PcapHelper classes to create a .tr trace file and a .pcap capturing file. In the fifth.cc script both the cwnd and RxDrop values are saved in a .dat file that can be used for further processes, while the sixth.cc script records these values in different files. The old value and new value of cwnd are recorded in an Ascii file, while the moment of dropping packets is recorded in a pcap file.

Method III) Automatic data collection and automatic plotting

Data collection and creation of plot files can be done automatically through FileHepler and GnuPlotHelper, respectively. As an example, the seventh.cc script uses GnuplotHelper to create a plot file and FileHelper to byte-count of each node's output packets. In this case, data collection and the plot file creation can be done automatically.

Running a code with different attributes

In general, there are two ways to change the value of variables if you want to run a program with new values. For example, *MaxPackets* is set to '1' in the first.cc script, meaning only one packet is sent through the client. Imagine you want to run this code in which the client sends 4 packets. Although you can edit the code and re-run it, as the easiest way, you can change each attribute through command line arguments.

You can check the possible command-line options of a code through the following command.

$ ./ns3 run "/scratch/first --PrintHelp"

This command returns "General Arguments" and "Program Options" of the first.cc script. General arguments refer to arguments of built-in classes and functions of ns3. For instance, "DataRate" is one of the built-in attributes of PointToPointNetDevice which is set to "5Mbps"

in this example. You can see the default values of this attribute and other attributes of this class through the following command.

$ ./ns3 run “scratch/first --PrintAttributes=ns3::PointToPointNetDevice”

You can run the following command if you want to run the first.cc script with DataRate=10Mbs without editing the code. You can assign a value for an attribute of a class even if you have not defined a value for it in your code.

$ ./ns3 run “scratch/first --ns3::PointToPointNetDevice::DataRate=10Mbps”

Program options refer to arguments and variables that you have defined in your code. These variables are defined at the beginning of the main function between “CommandLine……” and “cmd.Pa…..” lines. Applying --PrintHelp argument on the first.cc script does not return any output as program options because there is no variable defined in the corresponding part, while this argument returns *nCsma* and *verbose* variables for the second.cc, meaning you can change them through arguments of the run command as follows.

$ ./ns3 run “scratch/second --nCsma=10”

It means that the number of nodes in the LAN part is set to 11 and run it because there is one shared node with the point-to-point part. Similarly, applying --PrintHelp argument to the third.cc script clarifies that the value of *tracing* variable is *false* and the capturing process is disabled.

```
Program Options:
    --nCsma:    Number of "extra" CSMA nodes/devices [3]
    --nWifi:    Number of wifi STA devices [3]
    --verbose:  Tell echo applications to log if true [true]
    --tracing:  Enable pcap tracing [false]

General Arguments:
    --PrintGlobals:              Print the list of globals.
    --PrintGroups:               Print the list of groups.
    --PrintGroup=[group]:        Print all TypeIds of group.
    --PrintTypeIds:              Print all TypeIds.
    --PrintAttributes=[typeid]:  Print all attributes of typeid.
    --PrintVersion:              Print the ns-3 version.
    --PrintHelp:                 Print this help message.
```

Therefore, you should change it to true as follows if you want to have a .pcap trace file.

$ ./ns3 run “scratch/third --tracing=1”

For example, if you want to run first.cc script with a different number of packets that can be determined by the user, according to the provided explanations, you need to modify the code as follows to define *nPackets* as a program options variable.

```
int
main(int argc, char* argv[])
{
    uint32_t nPackets=1;
    CommandLine cmd(__FILE__);
    cmd.AddValue ("nPackets", "Number of client's packets", nPackets);
    cmd.Parse(argc, argv);
```

```
UdpEchoClientHelper echoClient(interfaces.GetAddress(1), 9);
echoClient.SetAttribute("MaxPackets", UintegerValue(nPackets));
```

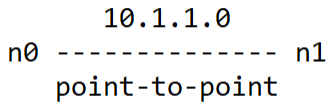
Project Tasks

Each task has different deliverables. Please follow the instructions carefully to cover all requested items.

Task 1: Review the scripts (30 points)

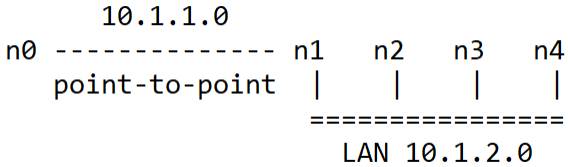
In this task, we just want to review first.cc, second.cc, and third.cc scripts. You should cover this task just in the demo file. You do not need to explain this task in your report.

- **First Script:** This script simulates a point-to-point network including two nodes, namely n0 as a client and n1 as a server, as follows.



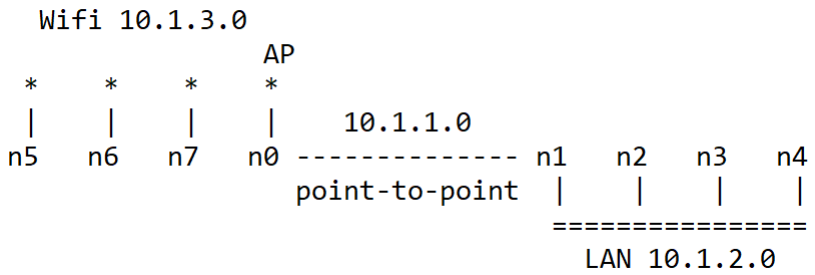
  a) Modifications on code: Select a random base address and select a random number smaller than 65000 as the port number of the server and apply them on the first.cc script.
  b) Run the first.cc and explain the simulation results (2 points).

- **Second Script:** This script simulates a network consisting of a point-to-point network and a LAN network. Point-to-point part includes two nodes, i.e., n0 as a client and n1 as a shared node between two networks. The LAN part consists of a shared node (n1) with P2P part and three extra nodes, which node n4 is the server of the network.



c) Modifications on code: Do modification on second.cc script to create two trace files entitled "<YourUBIT-P2P>.tr" for P2P part and "<YourUBIT-CSMA>.tr" for LAN part. Make changes to the code so that the user can define the number of packets (NPackets) through the run command arguments.
d) Run the second.cc with the following values as command arguments, PacketSize = 2048, NPackets = 3, Delay = 15ms, and 10 nodes in the LAN part. You should define all these parameters in one run command (4 points).
e) Open one of the generated trace files through a text editor tool and explain it in a few sentences (2 points).
f) Open "<YourUBIT-CSMA>.tr" through the TraceMetrics and explain the obtained simulation results (4 points).

- **Third Script:** This script adds a wireless network to the simulated scenario in the second script. Node n0 is the shared node between the wireless part and P2P part, this node plays the access point role in the wireless network, and node n7 is the client node in this scenario.



g) Modifications on code: Do modification on third.cc script to create a .xml file entitled "<YourUBIT>.xml", and three .pcap files entitled "<YourUBIT-P2P>.pcap" for P2P nodes, "<YourUBIT-wireless>.pcap" for Wi-Fi nodes and "<YourUBIT-CSMA>.pcap" for LAN nodes. Make changes to the code so that the user can define the number of packets (NPackets) through the run command arguments. Change the number of Wi-Fi nodes to 5 and the number of extra LAN nodes to 5.
h) Run the third.cc with NPackets = 4.
i) Open the created xml file through NetAnim and explain the created animation (6 points).
j) Open three pcap files, including one of the "<YourUBIT-P2P>.pcap", "<YourUBIT-wireless>.pcap", and "<YourUBIT-CSMA>.pcap" through Wireshark and explain them (6 points).
k) Open three pcap files, including one of the "<YourUBIT-P2P>.pcap", "<YourUBIT-wireless>.pcap", and "<YourUBIT-CSMA>.pcap" through Tcpdump and explain printer messaged. Are there any differences between Wireshark analysis and Tcpdump analysis (6 points)?

Task 2: Analysis of second and third scripts (22 points)

In this task, we want to run second.cc and third.cc scripts with different parameters and plot the graphs manually through Gnuplot. You should cover this task just in the report file. You do not need to explain this task in the demo file.

- **Plotting manually**
  a) Run second.cc and third.cc scripts with different numbers of extra nodes in the LAN part in the range of 5 to 10, extract the throughput of each run from the trace files, create "<YourUBIT_1>.txt" and "<YourUBIT_1>.plt" files manually, and plot a line graph manually through Gnuplot. This line graph should represent the throughput of LAN parts of second.cc and third.cc scripts at different numbers of nodes in one figure. Attach a screenshot of "<YourUBIT_1>.txt" and "<YourUBIT_1>.plt", the resultant chart in your report, and explain your figure in a few sentences (8 points).
  b) Run second.cc script with different numbers of packets in the range of 5 to 10, extract number of sent packets and number of received packets of each execution from the trace files, create "<YourUBIT_2>.data" and "<YourUBIT_2>.txt" files manually and plot a histogram. This histogram should represent the number of sent packets and number of received packets of LAN part of second.cc script at different numbers of packets in one figure. Attach a screenshot of "<YourUBIT_2>.data" and "<YourUBIT_2>.txt", the resultant figure in your report, and explain your figure in a few sentences (8 points).
- **Recording the position of a mobile node during the simulation**
  c) According to the explanations provided on page 94 of the tutorial file, add the required functions on third.cc to show the positions of node #5 during the simulation. Attach a screenshot of the output and explain it in a few sentences (6 points).

Task 3: TCP Congestion window size analysis through gnuplot-generated charts (24 points)

In this task, we want to run fifth.cc script with different parameters, do the data collection automatically, and plot graphs through gnuplot manually. You should cover this task only in the report file. You do not need to explain this task in your demo file.

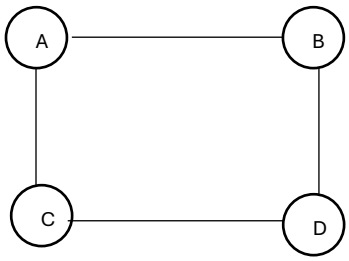
- **Cwnd Size Analysis**
  a) Run the fifth.cc script with various parameters given in the following table, create four .dat files, and attach a screenshot of a few lines of each file (4 points).
  b) Plot a congestion window size in terms of time through terminal command, attach the plots in your report, and answer the following questions (4 points).

Scenario	Delay	P2P Data rate	Error rate	Data size	Data generation rate	Generated file name
#1	Original	Original	Original	Original	Original	Your UBIT-1.dat
#2	Original	Original	10^{-10}	2048	Original	Your UBIT-2.dat
#3	5ms	Original	Original	Original	5Mbps	Your UBIT-3.dat
#4	5ms	6Mbps	10^{-3}	1024	4Mbps	Your UBIT-4.dat

c) In there any differences in the figures? Explain in a few sentences (2 points).
d) On each plot, identify the time of slow start, congestion avoidance, and fast recovery, and explain in your report (4 points).
e) Describe one of the .dat file in a few sentences (4 points).
f) Which scenario has a higher packet drop rate? Explain? (6 points)

Task 4: Simulating a point-to-point network (24 points)

In this task, we want to simulate the following point-to-point network consisting of four nodes. This task should be presented in the demo video.



Simulate this network according to the given simulation parameters.

Simulation Parameter	Value
Number of nodes	4
Link's Data rate	35Mbps
Link's Delay	27ms
IP between A and B	88.77.66.0, 255.255.255.0
IP between A and C	88.77.77.0, 255.255.255.0
IP between C and D	88.77.88.0, 255.255.255.0
IP between B and D	88.77.99.0, 255.255.255.0

Create a continuous flow/traffic between node A and node B, starting at 2.0 seconds, packet size = 1500 bytes, data rate = 700kbps, and stop at 8.0 seconds. This means that node A starts generating and continuously sending packets to node B at 2.0 seconds and stops at 8.0

seconds. There are several methods to generate continuous traffic such as OnOffHelper. This class has been utilized in several examples of ns3 scripts.

Note: You can use *grep* command to find a specific expression in all examples of a directory. For example, the following command returns all examples inside the ~/ns-allinone-3.43/ directory that include OnOffHelper. See pages 115 and 116 of the tutorial file for more details.

```
shahram@shahram:~/ns-allinone-3.43$ find . -name '*.cc' |xargs grep -i OnOffHelper
```

You can write a snippet like the following code to generate continuous traffic on the client side if you want to use OnOffHelper.

```
// Client for traffic generation
 OnOffHelper <name1> ("ns3::UdpSocketFactory",
                      Address (InetSocketAddress (<interfacenumber>.GetAddress (<client number>), <port number>)));
 <name1>.SetConstantRate (DataRate ("value"));
 <name1>.SetAttribute ("PacketSize", UintegerValue ("value"));
 ApplicationContainer <name2> = <name1>.Install (<client node>);
 <name2>.Start (Seconds (value);
 <name2>.Stop (Seconds (value));
```

Also, like the following code on the sink side (server side).

```
  // sink to receive the packets
 PacketSinkHelper <name3> ("ns3::UdpSocketFactory",
                           Address (InetSocketAddress (Ipv4Address::GetAny (), <port number>)));
 <name2> = <name3>.Install (<server number>);
 <name2>.Start (Seconds (value));
 <name2>.Stop (Seconds (value));
```

Moreover, create a continuous flow/traffic between node A and node D, starting at 4.0 seconds, packet size = 15500 bytes, data rate = 600kbps, and stop at 9.0 seconds. You can write a snippet like the following code on the client and server sides for remote indirect connections.

```
  // Traffic generator (remote)
 <name1>.SetAttribute ("Remote",
                       AddressValue (InetSocketAddress (<interfacenumber>.<clientnumber>, <port number>)));
 <name1>.SetConstantRate (DataRate ("value"));
 <name1>.SetAttribute ("PacketSize", UintegerValue ("value"));
 apps = <name1>.Install (<client number);
 <name2>.Start (Seconds (value));
 <name2>.Stop (Seconds (value));

 // Sink to receive the packets
 <name2> = <name3>.Install (<server number>);
 <name2>.Start (Seconds (value));
 <name2>.Stop (Seconds (value));
```

a) Create a animation file “<YourUBIT_3>.xml” and a trace file “<YourUBIT_3>.tr”, open them with the relevant tools, and explain you code and the obtained results in demo file (12 points).

b) Attach your code to your report and submit its source code along with other deliverables to UBLearns (12 points).

Bonus Task: (10 points)

In this task, we want to add automatic data collection to task 4 and plot two charts through Gnuplot similar to the seventh.cc script. You should cover this task only in the report file. You do not need to explain this task in your demo file.

Consider 10^{-6} and 10^{-8}, respectively, for the error rate for node B and node D. All nodes can have only IPv4 addresses and there are two flows in the network as explained above. Therefore, you need to record the packet-byte-counts in two different files.

a) Plot two packet-byte-count vs. time for flow #1 between node A and node B and flow#2 between node A and node C. Attach them in your report and explain each of them in a few sentences (5 points).
b) Attach your code to your report and submit its source code along with other deliverables to UBLearns (5 points).